

A generation algorithm for f-structure representations

Toni Tuells

IULA , Universitat Pompeu Fabra

La Rambla 30-32

Barcelona 08028, Catalonia (Spain)

tuells@upf.es

Abstract

This paper shows that previously reported generation algorithms run into problems when dealing with f-structure representations. A generation algorithm that is suitable for this type of representations is presented: the Semantic Kernel Generation (SKG) algorithm. The SKG method has the same processing strategy as the Semantic Head Driven generation (SHDG) algorithm and relies on the assumption that it is possible to compute the Semantic Kernel (SK) and non Semantic Kernel (Non-SK) information for each input structure.

CAT:	v
LEX:	generate
SUBCAT:	$\langle \text{NP}_{\boxed{1}}, \text{NP}_{\boxed{2}} \rangle$
SEM:	$\left[\begin{array}{l} \text{PRED: generate} \\ \text{ARG1: } \boxed{1} \\ \text{ARG2: } \boxed{2} \end{array} \right]$

CAT:	n
LEX:	sentence
SEM:	$\left[\text{REL: sentence} \right]$

Figure 1: Lexical entries for *sentence* and *program*

1 Introduction

In this paper we take up the problem of (tactically) generating a string from an f-structure representation; we will show that generation algorithms that have already been described in the literature are not directly applicable to this type of representations, and we will propose a new generation algorithm: the Semantic Kernel Generation (SKG) algorithm. We will also show that since the SKG generator is guided by the *semantic-head* and *syntactic-head* relations, it can be seen both as a variant of the *semantic head driven generation algorithm* (SHDG) (Shieber *et al.* 1990) and the *syntactic-head driven generation algorithm* (SynHDG) (König 1994). A former version of this work was originally reported in (Nicolov *et al.* 1996), which also describes alternative generation algorithms for f-structure representations.

2 The Semantic Kernel Generation Algorithm

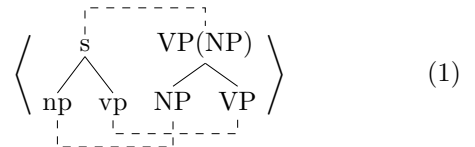
In order to show why former generators fail to generate from f-structures and why SKG works, we will assume the grammar fragment and lexical entries (originally described in (Nicolov *et al.* 1996)) which are shown in Figures 1, 2.¹ As for the grammar fragment, note that *rules 1a and 1b* introduce modifiers at sentence level, and *rule 3* introduces modifiers at vp level. *Rule 2* combines the subject with the *vp*. *Rule 4* deals with the complements of a *vp*.

The analysis of the sentence *The little prolog program generated the complex sentence quickly* is shown in Figure 3.² Note that input semantics represents the deep predicate argument structure of sentences for the

generator; modifiers are contained in set-valued feature “mod”.

First we look at the results we obtain after applying former generation methods on f-structure representations and then we describe the SKG algorithm.

For expository purposes we will use the graphical notation used in (König 1994) to describe the generation algorithms. Following (König 1994): we will assume that the syntax-semantics-relation for a given grammar is stated by pairs of trees. The left tree shows a local syntactic dependency, whereas the right tree defines a local semantic dependency. We also assume that there is a one-to-one mapping from the nonterminal leaf nodes of the syntactic tree to the leaf nodes of the local semantic tree. Note that this is only a graphical notation for the *rule-to-rule hypothesis*, i.e., the fact that in the grammar each syntactic rule is related to a semantic analysis rule. An example is given below:



The head-corner generator ((Noord 1993), a variant of SHDG) and SynHDG are (graphically) described in Figure 4 (taken directly from (König 1994)). The *lex* rule is the *prediction step* of the algorithm, i.e. it restricts the selection of lexical entries to those that can be linked with the local goal (visualized by a dotted line). The *hc_complete* rule is the *bottom-up step* which selects a rule for which *xh* is the syntactic head and *X_h* is the semantic head. As a result, it also predicts the head's sisters, which have to be expanded recur-

¹For the grammar fragment only the relevant semantic information is shown.

²The example is due to Nicolas Nicolov.

$$\begin{aligned}
(1a) \quad & \begin{bmatrix} \text{cat:} & s \\ \text{sem:} & \text{SEM} \\ \text{sem:mod:} & [M \mid \text{MODS}] \end{bmatrix} \longrightarrow \begin{bmatrix} \text{cat:} & \text{adv} \\ \text{sem:} & M \end{bmatrix}, \begin{bmatrix} \text{cat:} & s \\ \text{sem:} & \text{SEM} \\ \text{sem:mod:} & \text{MODS} \end{bmatrix} \\
(1b) \quad & \begin{bmatrix} \text{cat:} & s \\ \text{sem:} & \text{SEM} \\ \text{sem:mod:} & [M \mid \text{MODS}] \end{bmatrix} \longrightarrow \begin{bmatrix} \text{cat:} & s \\ \text{sem:} & \text{SEM} \\ \text{sem:mod:} & \text{MODS} \end{bmatrix}, \begin{bmatrix} \text{cat:} & \text{adv} \\ \text{sem:} & M \end{bmatrix} \\
(2) \quad & \begin{bmatrix} \text{cat:} & s \\ \text{sem:} & \text{SEM} \end{bmatrix} \longrightarrow \begin{bmatrix} \text{cat:} & \text{np} \\ \text{sem:} & \text{SEM_SUBJ} \end{bmatrix}, \begin{bmatrix} \text{cat:} & \text{vp} \\ \text{sem:} & \text{SEM} \\ \text{subcat:} & \begin{bmatrix} \text{cat:} & \text{np} \\ \text{sem:} & \text{SEM_SUBJ} \end{bmatrix} \end{bmatrix} \\
(3) \quad & \begin{bmatrix} \text{cat:} & \text{vp} \\ \text{sem:} & \text{SEM} \\ \text{sem:mod:} & [\text{MOD} \mid \text{MODS}] \end{bmatrix} \longrightarrow \begin{bmatrix} \text{cat:} & \text{adv} \\ \text{sem:} & \text{MOD} \end{bmatrix}, \begin{bmatrix} \text{cat:} & \text{vp} \\ \text{sem:} & \text{SEM} \\ \text{sem:mod:} & \text{MODS} \end{bmatrix} \\
(4) \quad & \begin{bmatrix} \text{cat:} & \text{vp} \\ \text{sem:} & \text{SEM} \\ \text{subcat:} & [\text{SUBJ} \mid \text{REST}] \end{bmatrix} \longrightarrow \begin{bmatrix} \text{cat:} & \text{vp} \\ \text{sem:} & \text{SEM} \\ \text{subcat:} & [\text{SUBJ}, \begin{bmatrix} \text{cat:} & X \\ \text{sem:} & \text{SEM}_X \end{bmatrix} \mid \text{REST}] \end{bmatrix}, \begin{bmatrix} \text{cat:} & X \\ \text{sem:} & \text{SEM}_X \end{bmatrix} \\
(5) \quad & \begin{bmatrix} \text{cat:} & \text{vp} \\ \text{sem:} & \text{SEM} \\ \text{subcat:} & \text{SUBCAT} \end{bmatrix} \longrightarrow \begin{bmatrix} \text{cat:} & v \\ \text{sem:} & \text{SEM} \\ \text{subcat:} & \text{SUBCAT} \end{bmatrix} \\
(6) \quad & \begin{bmatrix} \text{cat:} & \text{np} \\ \text{sem:} & \text{SEM} \end{bmatrix} \longrightarrow \begin{bmatrix} \text{cat:} & \text{det} \end{bmatrix}, \begin{bmatrix} \text{cat:} & \text{n2} \\ \text{sem:} & \text{SEM} \end{bmatrix} \\
(7) \quad & \begin{bmatrix} \text{cat:} & \text{n2} \\ \text{sem:} & \text{SEM} \end{bmatrix} \longrightarrow \begin{bmatrix} \text{cat:} & n \\ \text{sem:} & \text{SEM} \end{bmatrix} \\
(8) \quad & \begin{bmatrix} \text{cat:} & \text{n2} \\ \text{sem:} & \text{SEM} \\ \text{sem:mod:} & [\text{MOD} \mid \text{MODS}] \end{bmatrix} \longrightarrow \begin{bmatrix} \text{cat:} & \text{adj} \\ \text{sem:} & \text{SEM} \end{bmatrix}, \begin{bmatrix} \text{cat:} & \text{n2} \\ \text{sem:} & \text{SEM} \\ \text{sem:mod:} & \text{MODS} \end{bmatrix}
\end{aligned}$$

Figure 2: Grammar Fragment. Only semantic information is shown

$$\begin{bmatrix} \text{MOD:} & \langle \text{quick} \rangle \\ \text{PRED:} & \text{generate} \\ \text{ARG1:} & \begin{bmatrix} \text{DEF:} & + \\ \text{MOD:} & \langle \text{little, prolog} \rangle \\ \text{REL:} & \text{program} \end{bmatrix} \\ \text{ARG2:} & \begin{bmatrix} \text{DEF:} & + \\ \text{MOD:} & \langle \text{complex} \rangle \\ \text{REL:} & \text{sentence} \end{bmatrix} \end{bmatrix}$$

Figure 3: Input semantics

sively (*top-down prediction*). The difference between SHDG and SynHDG is the link relation for semantic structures: in (Noord 1993) the *semantic-based link relation* is defined as follows:

$$\text{link}(X, X_i) \text{ if} \quad (2)$$

X and X_i are *identical*. If we represent semantics using first order terms, then we only have to check whether X and X_i unify. As for the SynHDG algorithm, the *semantic-based link relation* is defined as follows (König 1994):

$$\text{link}(X, X_i) \text{ if} \quad (3)$$

X_i is a substructure of X . In practical terms, X_i is an element of the bag of semantic keywords that constitute X (König 1995).

2.1 The direct application of former generation procedures to f-structure representations

We will illustrate the problems SHDG (more specifically, a variant of it: the head-corner generator described in (Noord 1993)) runs into when dealing with f-structures by following its application to the input semantics given below (which corresponds to the np *the complex sentence*):

$$\left[\begin{array}{ll} \text{cat:} & \text{np} \\ \text{sem:} & \left[\begin{array}{ll} \text{rel:} & \text{sentence} \\ \text{def:} & + \\ \text{mod:} & [\text{complex}] \end{array} \right] \end{array} \right] \quad (4)$$

According to the algorithm described in Figure 4, and due to the *syntactic-head* and *semantic-head link relation*, the *lex* rule can only be applied to the lexical entry for *sentence* (Figure 1). However, since the semantic link relation is defined in terms of *unification*, applying rule *lex* leads to a new semantic goal which is *identical* to the input semantics. Next we need to apply to apply the *hc_complete* rule; rule 7 is the only possible candidate. After applying it, our current goal is the following:

$$\left[\begin{array}{ll} \text{cat:} & \text{n2} \\ \text{sem:} & \left[\begin{array}{ll} \text{rel:} & \text{sentence} \\ \text{def:} & + \\ \text{mod:} & [\text{complex}] \end{array} \right] \end{array} \right] \quad (5)$$

At this point, a new *hc_complete* step needs to be taken. Now we have two candidates: rules 6 and 8. If we select rule 6, and after generating recursively the determiner, we end up having generated only part of the sentence: *the sentence*. Rule 8, in its turn, can be *always* selected; consequently, we could end up having semantic goals that would look like that:

$$\left[\begin{array}{ll} \text{cat:} & \text{n2} \\ \text{sem:} & \left[\begin{array}{ll} \text{rel:} & \text{sentence} \\ \text{def:} & + \\ \text{mod:} & [X, \dots | \text{complex}] \end{array} \right] \end{array} \right] \quad (6)$$

In other words, *the generator would loop and would not terminate*.

The problems discussed above lead us to the following conclusion: the SHDG generator is *neither complete nor coherent*. These issues also arise with first order terms (see discussion in (Shieber *et al.* 1990)); the problem here is that we lack the definition of grounded feature structures.

2.2 Semantic Kernel Generator

The main assumption behind the SKG algorithm is that the generator is capable of distinguishing between the following types of semantic information within input structures:

- **Semantic Kernel (SK) Information:** Semantic structure completely which is predictable from the lexicon (i.e, there is at least one lexical entry which subsumes this structure).
- **Non Semantic Kernel (Non-SK) Information:** Semantic structure which is not predictable from any lexical entry (typically, lists). In our grammar, modifiers are represented as a list. This list is Non-SK information.

Similarly, the generator is given the following information with respect to the types of rules:

- **SK Rules:** Rules which do not add Non-SK information.
- **Non-SK Rules:** Rules which add Non-SK information. Rules 1,3,8 in our grammar.

The hypothesis behind this classification is that of *structural predictability*. SK information comes from the lexicon (i.e, SK information can be seen as grounded feature terms), and non SK information is introduced through rules. In other words, the generator knows whether each type of input structure comes from a lexical entry or whether it has been constructed from a (non SK) rule. Thus, the restrictedness of the algorithm is due to the fact that it operates under the assumption that we can recursively decompose each input structure into SK and Non-SK information.

A graphical version of the SKG algorithm is given in Figure 5.

In sum, this is the information the generator needs to know about the grammar:

- link relation (head relation).
- The SK and Non-SK substructures of a given semantic representation.
- The distinction between SK rules and Non-SK rules.
- The syntactic goals to generate SK and Non-SK information.
- The syntactic goal we obtain after combining SK and Non-SK information.

In order to show how the SKG algorithm works we will follow its application to the input semantics for *the complex sentence* given in example 4. For this input semantics, we have two SK structures:

$$\left[\begin{array}{ll} \text{rel:} & \text{sentence} \end{array} \right] \quad (7)$$

$$\left[\begin{array}{ll} \text{def:} & + \end{array} \right] \quad (8)$$

all leaves are labeled with terminals and the tree does not contain any dotted lines (*global-success*)

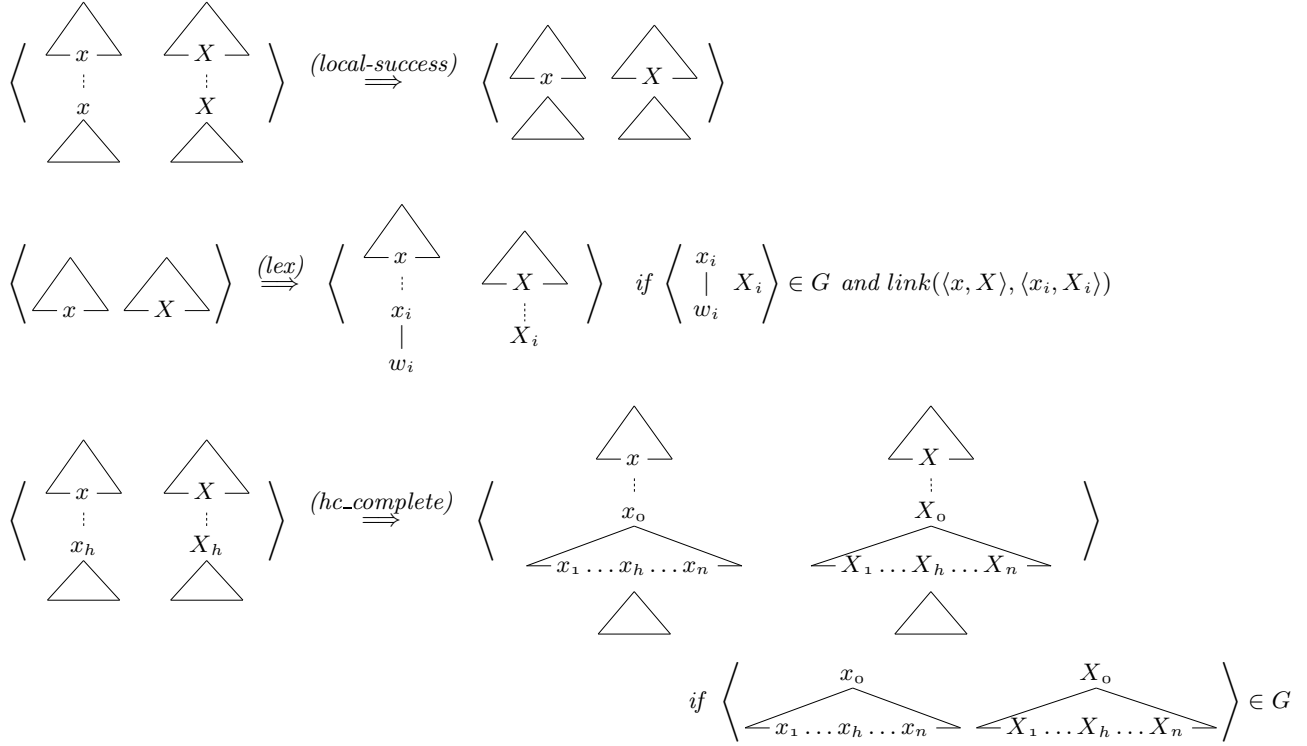


Figure 4: Head-Corner Generator (G grammar description; x_i syntactic category; X_i semantic representation) (Taken from (König 1994))

and one nonSK structure:

$$\left[\begin{array}{l} \text{mods: } [\text{complex}] \end{array} \right] \quad (9)$$

The *lex* rule cannot be applied because of the SK structure condition: input semantics has nonSK information. Thus, we can only apply the second *hc-corner step*. This forces us to start from *rule 6* and generate (top-down) the following goals:

$$\left[\begin{array}{l} \text{cat: } \text{det} \\ \text{sem: } [\text{def: } +] \end{array} \right] \quad (10)$$

$$\left[\begin{array}{l} \text{cat: } \text{n2} \\ \text{sem: } \left[\begin{array}{l} \text{rel: } \text{sentence} \\ \text{mods: } [\text{complex}] \end{array} \right] \end{array} \right] \quad (11)$$

Note that the generator has been told about the relation between nonSK information and SK and nonSK rules, therefore it knows where the modifiers come from. The determiners generation is reduced to applying the *lex* rule. To generate the *n2* goal (example 11) we proceed as before; this goal has nonSK information, so the generator starts from *rule 8* and generates (top-down) the appropriate subgoals:

$$\left[\begin{array}{l} \text{cat: } \text{n2} \\ \text{sem: } [\text{rel: } \text{sentence}] \end{array} \right] \quad (12)$$

$$\left[\begin{array}{l} \text{cat: } \text{adj} \\ \text{sem: } [\text{rel: } \text{complex}] \end{array} \right] \quad (13)$$

The generation of the subgoals above is straightforward, since they do not contain nonSK information and the *lex* rule can be applied without problems. The application of the *hc-corner step* to each of the subgoals deserves further comments, since we are running the risk of having *termination* problems. For example, once we have applied the *lex* rule and the *hc-corner step* for *sentence* we obtain the goal in example 12. One may wonder whether we could apply *rule 8* again and end up having subgoals like the following:

$$\left[\begin{array}{l} \text{cat: } \text{n2} \\ \text{sem: } \left[\begin{array}{l} \text{rel: } \text{sentence} \\ \text{mods: } [\text{X}, \dots] \end{array} \right] \end{array} \right] \quad (14)$$

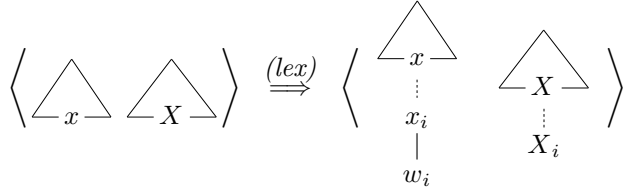
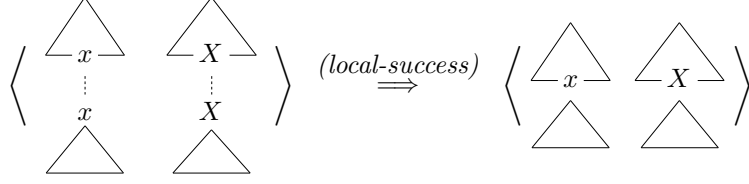
This situation is not possible, since if we only have SK information only SK rules can be applied, and *rule 8* is a nonSK rule (see conditions on *hc-complete step* (1) in Figure 5).

Another example will clarify how the generator works. Assuming we want to generate a string for the semantic representation in Figure 3, the following sentences should be generated according to the grammar:³

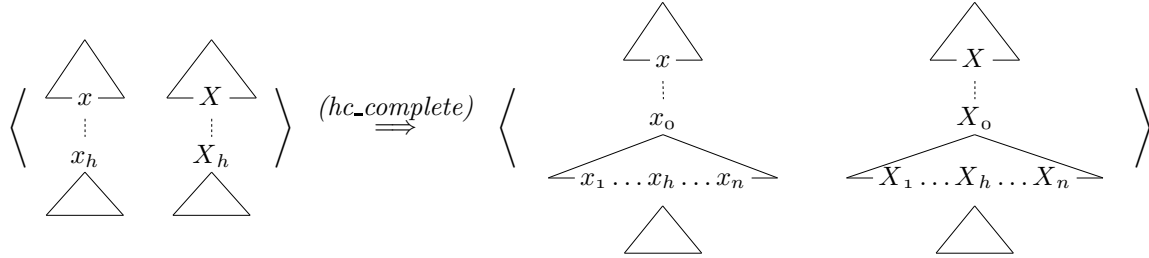
- the {little,prolog} program generated the complex sentence quickly.

³In this example we will only concentrate in the generation of *quickly* at sentence or vp level; the rest of modifiers (*complex*, *little*, *prolog*) for the np level would be generated in an identical manner.

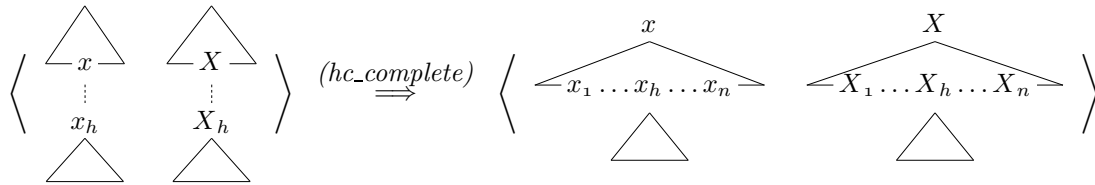
all leaves are labeled with terminals and the tree does not contain any dotted lines (*global-success*)



if $\left\langle \begin{array}{c} x_i \\ | \\ w_i \end{array} \middle| X_i \right\rangle \in G$ and $link(x, x_i)$ and $sk(X, X_i)$ and $SK(X)$



if $\left\langle \begin{array}{c} x_o \\ \swarrow \quad \searrow \\ x_1 \dots x_h \dots x_n \end{array} \quad \begin{array}{c} X_o \\ \swarrow \quad \searrow \\ X_1 \dots X_h \dots X_n \end{array} \right\rangle \in G_{SK}$ and $SK(X)$



if $\left\langle \begin{array}{c} x \\ \swarrow \quad \searrow \\ x_1 \dots x_h \dots x_n \end{array} \quad \begin{array}{c} X \\ \swarrow \quad \searrow \\ X_1 \dots X_h \dots X_n \end{array} \right\rangle \in G$ and $\neg SK(X)$ and $sk(X, X_h)$

Figure 5: Semantic Kernel Generator (G grammar description; x_i syntactic category; X_i semantic representation; $SK(X)$ is true if X contains only SK information; G_{SK} grammar description with only SK rules; $G_{\neg SK}$ grammar description with only nonSK rules; $sk(X, X_h)$ is true if X_h is SK information of X (Adapted from (König 1994))

- *quickly the {little,prolog} program generated the complex sentence.*
- *the {little,prolog} program quickly generated the complex sentence.*

The generator detects that the semantic representation consists of both a SK structure and a Non-SK structure (“quickly”). Thus, according to the grammar, there are several ways of generating these structures given the original goal (which is a sentence). The generator tries the following combinations:

- It generates a “S” string for the SK information and a “adv” string for the Non-SK information. Both strings can be combined in two ways (which corresponds to rules R1a and R1b). This gives us two of the possibilities.
- It generates a “VP” string for the SK information and “adv” string for the Non-SK information (this corresponds to rule R3). After generating these strings, and according to rule R2, we link the “VP” to “S”

3 Discussion

The SKG generator proceeds top-down, generating the appropriate subgoals, when it finds nonSK information. It proceeds bottom-up when lexical prediction can be made (when there is only SK information), and the *head-corner step* for SK information can only be performed using SK rules, thus avoiding *termination* problems.

The way our generator works and the distinction between SK and Non-SK information resembles the definition of the *restrictor* operator and the treatment of modifiers given in (Wedekind & Kaplan 1993). The difference is obviously the context of application: In (Wedekind & Kaplan 1993) the main interest is structural misalignment between f-structure and semantic representations, whereas our concern is string generation from f-structure representations.

4 Implementation

Our framework has been the Sicstus-Prolog version of the CUF language (Dörre & Dorna 1993) plus a layer on top of it which implements the grammar formalism, the (left-corner) parser and the SKG generator.

5 Conclusions

We have shown that for f-structure representations previously proposed generation algorithms run into problems, and therefore, we have presented an alternative: the SKG algorithm. The main assumptions behind the algorithm is the following: for each semantic input it is possible to compute its SK and Non-SK information. We have also shown that our approach resembles the definition of the *restrictor* operator and the treatment of modifiers given in (Wedekind & Kaplan 1993) to deal with structural misalignments between f-structure and semantic representations.

6 Acknowledgments

I would like to thank Esther König, Michael Dorna and Nicolas Nicolov for valuable comments on the ideas

presented in this paper, and to Cristina Corcoll for spellchecking it. Obviously, I am responsible for any mistake.

References

- (Dörre & Dorna 93) Dörre J. and Dorna M. 1993. CUF - A Formalism for Linguistic Knowledge Representation. Deliverable R.1.2A, DYANA, Institut für Maschinelle Sprachverarbeitung, University of Stuttgart, Germany.
- (König 94) Esther König. 1994. Syntactic-Head Driven Generation. In *Proceedings of Coling-94*, Kyoto, Japan.
- (König 95) Esther König. 1995. LexGram - a practical categorial grammar formalism. In *Proceedings of CLNLP-95*, Edinburgh, Scotland.
- (König 96) Esther König. 1996. Personal communication.
- (Nicolov *et al.* 96) N.Nicolov & C.Mellish and T.Tuells. Mixed Semantic-Syntactic Head Driven Generation for Constraint-based Grammars. 1996. Draft paper.
- (Noord 93) Gertjan van Noord. 1993. *Reversibility in Natural Language Processing*. PhD thesis, University of Utrecht.
- (Shieber *et al.* 90) Shieber S., van Noord G., Moore R. and Pereira F. 1990. Semantic-head-driven Generation. In *Computational Linguistics*, 16(1)
- (Wedekind & Kaplan 93) Wedekind, J. and Kaplan, R. 1993. Type-Driven Semantic Interpretation of f-structures. In *Proceedings of EACL-93*, Utrecht, The Netherlands.